# Evolutionary Design of a Robotic Material Defect Detection System

Gary Ballard
Dr. Tom Howsman
Mike Craft
Dynamic Concepts, Inc.
P.O. Box 97
Madison, AL 35758
email: gballard @ dynamic-concepts.com

Dan O'Neil
Dr. Jim Steincamp
NASA, Marshall Space Flight Center, AL 35812
email: dan.oneil @msfc.nasa.gov

## KEYWORDS

Evolutionary robots, neural network, genetic algorithm

## ABSTRACT

During the post-flight inspection of SSME engines, several inaccessible regions must be disassembled to inspect for defects such as cracks, scratches, gouges, etc. An improvement to the inspection process would be the design and development of very small robots capable of penetrating these inaccessible regions and detecting the defects. The goal of this research was to utilize an evolutionary design approach for the robotic detection of these types of defects. A simulation and visualization tool was developed prior to receiving the hardware as a development test bed. A small, commercial off-the-shelf (COTS) robot was selected from several candidates as the proof of concept robot. The basic approach to detect the defects was to utilize Cadmium Sulfide (CdS) sensors to detect changes in contrast of an illuminated surface. A neural network, optimally designed utilizing a genetic algorithm, was employed to detect the presence of the defects (cracks). By utilization of the COTS robot and CdS sensors, the research successfully demonstrated that an evolutionarily designed neural network can detect the presence of surface defects.

## INTRODUCTION

The research consisted of three distinct tasks including the selection of the commercial off-the-shelf (COTS) robot, the development of a simulation and visualization tool, and the evolutionary design of a robot capable of detecting defects utilizing a neural network and genetic algorithm. The remainder of this paper will discuss these tasks in detail, provide brief introductions to neural networks and genetic algorithms, and then provide the results of the design and overall conclusions.

## COTS ROBOT SELECTION

Several issues were considered in the selection of the mobile robot. The mobile robot had to be programmable and relatively inexpensive. Also, it was very desirable that the robot be as small as possible. A total of nine commercial robots were considered during the trade study. The PocketBot, shown in Figure 1, was selected from the nine robots.

Note the size of the PocketBot compared to the penny in Figure 1. The size of the PocketBot is 63 mm x 48 mm x 38 mm. The PocketBot uses a 9V battery and is capable of serial communication.

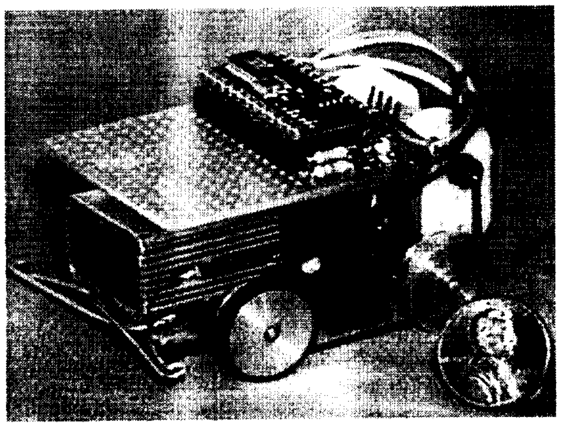


**Figure 1 PocketBot**

## SIMULATION AND VISUALIZATION TOOL

During the initial stages of the research a simulation and visualization tool was developed in order to test various robot designs and route planning. Figure 2 provides an illustration of the simulation and visualization tool

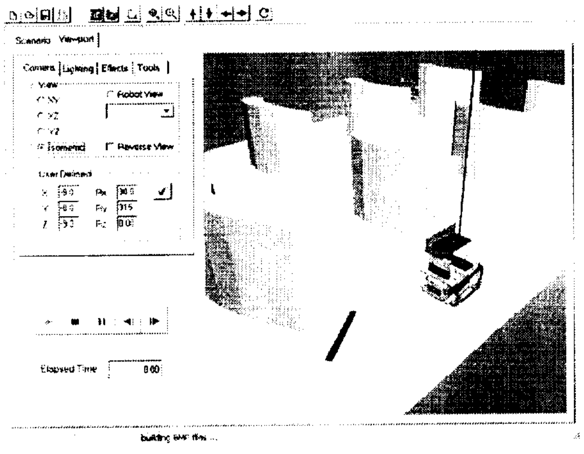


**Figure 2 Simulation and Visualization Tool**

The simulation and visualization tool consisted of a configurable virtual workspace with user selectable view and light source orientation. Various graphic effects such as perspective, object textures, and colors can be modified by the user. Standard visual tools, specifically zooming and panning, are available within the tool. The capability for command tool automation was also built into the visualization tool.

## EVOLUTIONARY DESIGN

The basic approach in the evolutionary design was to utilize photo-resistive CdS sensors to detect changes in contrast of an illuminated surface. Figure 3 provides a pictorial of the PocketBot after the necessary modifications were made to enable the detection of defects by the robot.

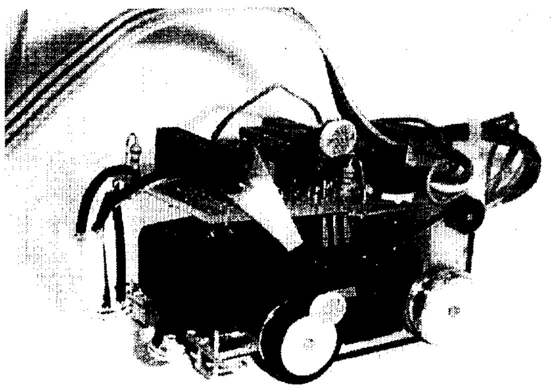


**Figure 3 Enhanced PocketBot**

The two LED illuminators are the white items on the front and side of the robot in Figure 3, which were wrapped with cellophane tape to diffuse the provided light source. One LED illuminator was utilized for track following and one LED illuminator was used to provide light for crack detection. A total of four CdS sensors were used on the final design of the robot. Two of these CdS sensors were used for path following and two were utilized for defect detection. Note that the two CdS sensors on the front of the PocketBot are the path following sensors while the two CdS sensors mounted on the side of the robot were used for crack detection on vertical surfaces. The white items on the back of the PocketBot are the wheel encoders. The serial cable pictured in Figure 3 is connected to the serial port of the robot and was used to communicate with the PocketBot.

A neural network was employed to detect the presence of the defects or cracks based upon the readings of the CdS sensors and the wheel encoder readings. A genetic algorithm was used to evolve the design of the neural network.

## Neural Network

Typically, neural networks must be trained. Training occurs by providing the network with a large number of inputs for which the corresponding output is known. In other words, the neural network designer will provide the network (i.e., the "student") with a host of problems, but the designer will also provide the answers.

The network is typically trained by a process known as "supervised learning" (i.e., a repetitive presentation of corresponding inputs and desired outputs). Network "learning" occurs by adjusting the input weights ($w_i$) in such a way as to increase the likelihood of getting the correct answer. The procedure (algorithm) utilized to modify the weights accounts for most of the differences between the various neural network models (e.g., back-propagation, radial basis, Hebbian learning, etc.).

At some point in the training process, an effective network must begin to generalize. This means that the network can successfully solve problems that it has not been exposed to during the training process. Simple memorization of the training data is not sufficient.

The neural network design chosen for this research is a feed-forward network and was trained using back propagation. As will be discussed in the next section, a genetic algorithm was utilized to optimize the number of input measurement records and hidden layer neurons used in the neural network. Figure 4 illustrates the neural network utilized in this research for defect or crack detection. The inputs consist of the low CdS sensor reading, the high CdS sensor reading, the difference between the sensor readings, the left encoder value, and the right encoder value. These inputs are provided to the network for the current time step, six previous time steps, and six future time steps. The genetic algorithm was utilized to optimize the number of past and future time steps utilized in the neural network. The genetic algorithm also 'selected' the optimal number of hidden layer neurons to be 179 for the final design.

One of the major obstacles during the neural network portion of the research was the generation of sufficient quantities of training data. A training course, shown in Figure 5, was designed specifically for the purpose of generating training data.

A typical set of training data is also pictured in Figure 5. The training data set on the left is a complete run and consists of top CdS sensor, bottom CdS sensor, left encoder, and right encoder measurements. The training data shown on the lower right in Figure 5 is the same data as provided on the lower left except that the x-axis has been restricted to the range of 240 to 320 seconds.

In order to generate the vast quantity of data required for training the neural network, the robot was repeatedly exposed to the training course and the CdS sensor and encoder readings were recorded. This data was then organized and the utilized in the training of the neural network.

Once the training data was generated, a target vector had to be constructed in order to train the neural network. The network was designed with two output neurons: a defect neuron and an orientation neuron. The target vector for the defect neuron was selected such that one equals a defect and zero corresponds to no defect. The orientation vector was chosen such that 1 corresponds to a 45 deg orientation, 0 equals a zero or vertical orientation, and $-1$ equals a negative 45 deg orientation.

After successful training of the neural network, the network would output a value close to one for the defect neuron when a crack or defect was encountered. The orientation neuron then allows the user to estimate the orientation of the defect by outputting the a value between $-1$ and $+1$ ($-1 = -45$ deg, $0 = 0$ deg, and $+1 = +45$ deg).

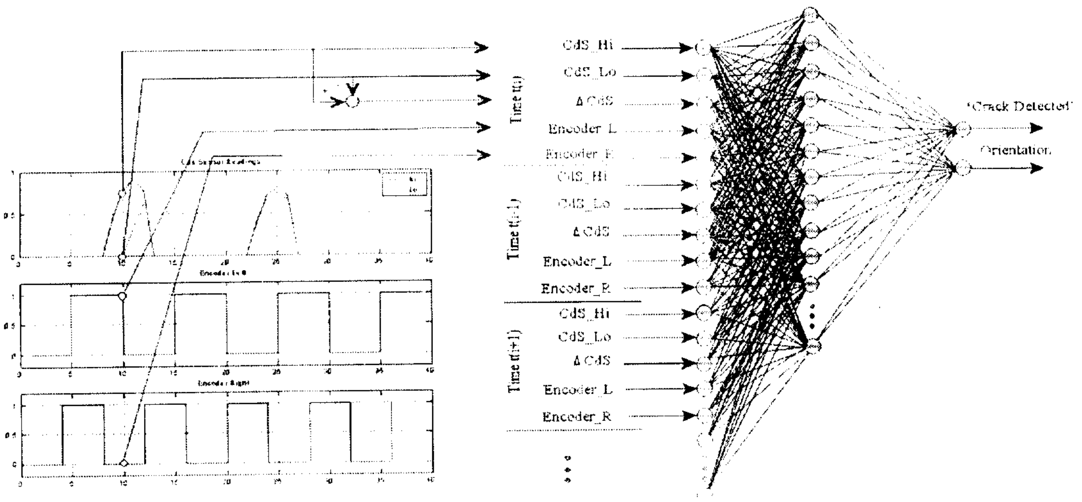


**Figure 4 Crack ID Neural Network Architecture**

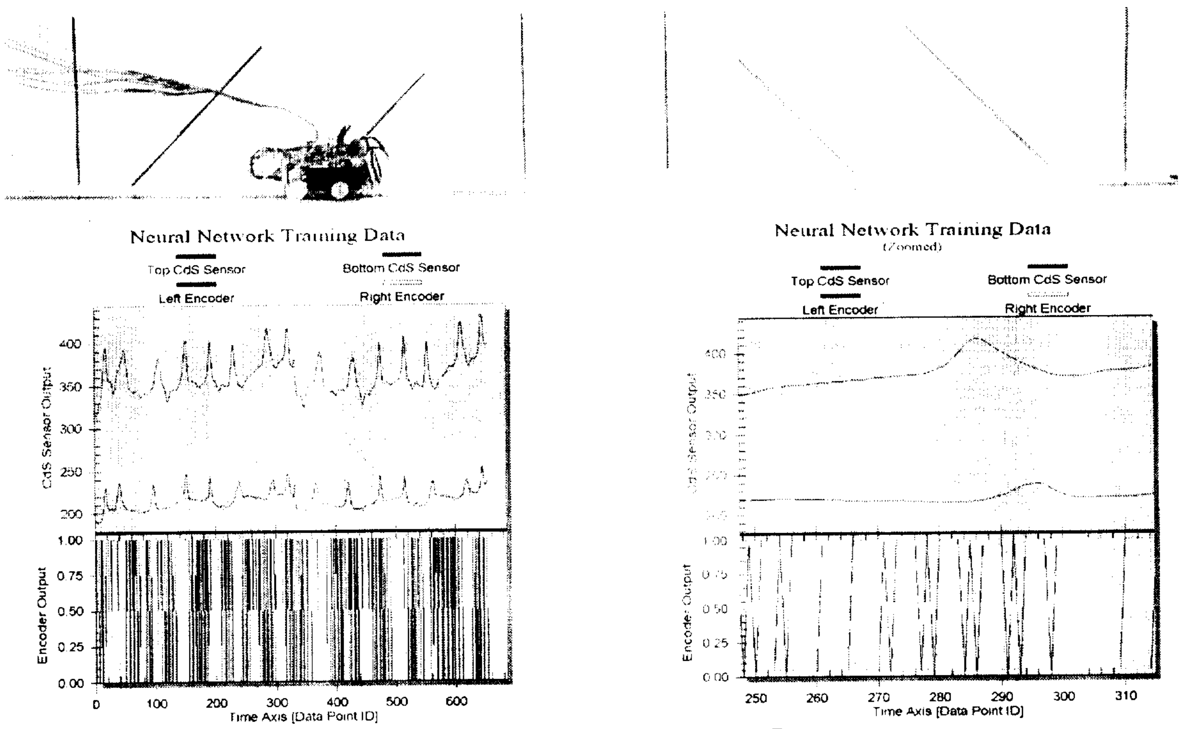


**Figure 5 Neural Network Training Data**

## Genetic Algorithm

Simply stated, a genetic algorithm is a computational search procedure that starts with a set of possible solutions, or members of the population, and iteratively modifies the population members in such a way as to minimize (or maximize) a particular fitness (error) index. The iterative, or evolutionary changes that occur in such a population are governed by rules which seek to emulate natural evolutionary theory, i.e., "natural selection". To form a new generation of solutions, the population members from the previous generation who perform the best (lowest error index) are carried forward for "reproduction". Each member of the population "competes" with every other member for the chance to reproduce. A simple flowchart of the genetic algorithm is shown in Figure 6. A new generation is formed in the darkened boxes.

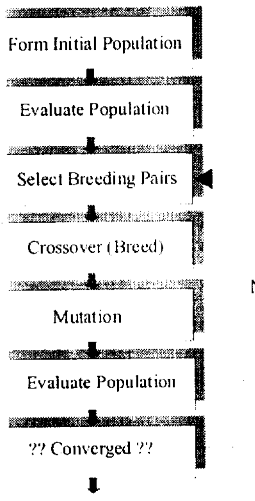


**Figure 6 Simple GA Flowchart**

In the design of a neural network for a particular application, it is typically not known *a priori* what the size and structure of the network should be (i.e., how large and complex to make the artificial "brain. A genetic algorithm (GA) can be used to "optimize" the size of the network. It was assumed that a two layer neural network would be employed, however the number of hidden layer neurons and the number of input data records to the network were optimized by the genetic algorithm.

The encoding scheme used for the genetic algorithm consisted of binary encoding on two variables. Eight bits, which allow 256 distinct values, were used to represent the number of hidden layer neurons. Three bits were utilized for the time records, which allows for 8 distinct choices (000 through 111). These eight choices are directly translated into an integer r with a range of 0 to 7. Additionally, the variable r may be converted to the number of past (and future) time records by the formula.

$$n\_p = 2*(r + 1)$$

The range of n_p is 2 through 16 where n_p must be an even number. The assumption was made that the same number of past time records as "future" time records would be used in the network (e.g., if we use data at $t_{i-1}$, then we use data at $t_{i+1}$. ). Note that this symmetry about the evaluation point ($t_i$) requires that the evaluation be delayed until a certain number (n_p) of future data samples are available.

The genetic algorithm will try to minimize the following function:

$$Obj(r, n\_hl) = 10^{(e\_nn/0.03)} * ( \ln(r + 1) + 1 ) * ( \log(n\_hl) + 1 )$$

Where e_nn is the neural network training error (~mean sq. error) and e_nn = e_nn(n_p, n_hl), and n_p = n_p(r). Note that n_hl represents the number of hidden layer neurons.

For each (r, n_hl) pair (which sizes the neural network), the network is trained for 600 epochs in an attempt to reach an error goal of 0.04. Many networks did not reach the training goal of 0.04. This particular objective function severely penalizes networks that are unable to reach the targeted training error goal of 0.04. The number of time records (~r) utilized by the network is penalized to a greater extent than the number of hidden layer neurons since maintaining a large amount of sensor data in RAM on the robot is difficult (The Parallax Stamp processor has only 26 bytes of RAM!).

The number of individuals in the population was chosen to be 10 while the maximum number of generations was set to 100. Elitism and linear fitness scaling and limitation were utilized in the genetic algorithm. The selection for reproduction was done with *Stochastic Universal Sampling*. Also, standard single-point crossover with a mutation rate of ~5% was utilized in the genetic algorithm.

The genetic algorithm selected r=2 and number of hidden layers equal to 179 as the "best" solution as illustrated in Figure 7. This corresponds to using 6 past, 1 current, and 6 "future" data sets in the NN evaluation. Note that the execution time for the GA on a single computer was approximately 60 hours

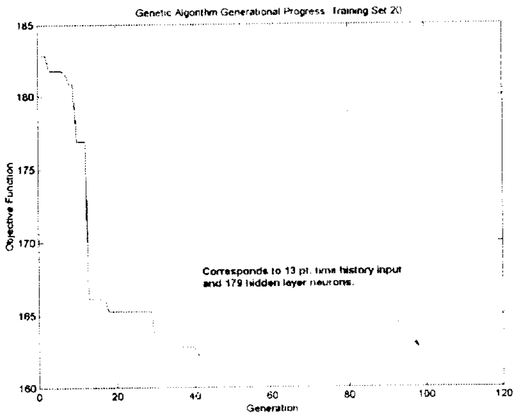
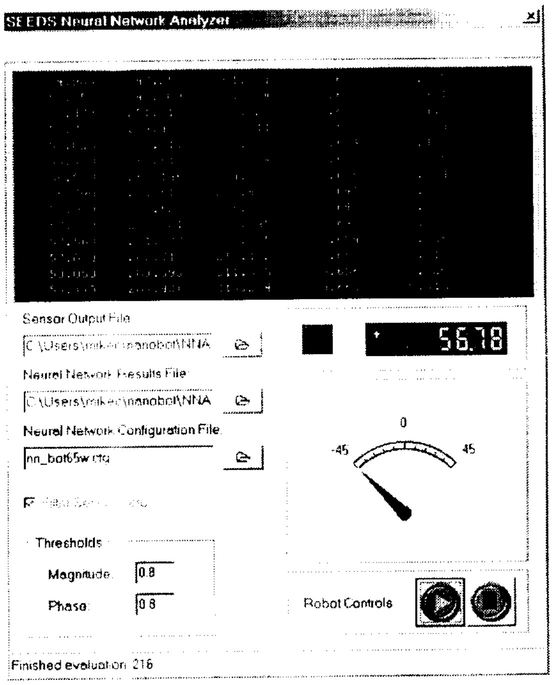


**Figure 7 GA Evaluation Results**

A Win32 application was developed which controls the serial i/o, displays the sensor output data, includes a low-pass filter, hosts the actual neural network, and indicates the presence of a crack and its orientation. Note that the sensor and encoder results from the robot were sent from the robot through the serial cable to this Win32 program, which used the neural network to evaluate these measurements. Figure 8 illustrates this Win32 application.


SEEDS Neural Network Analyzer

Sensor Output File: C:\Users\mike\nanobot\NNA

Neural Network Results File: C:\Users\mike\nanobot\NNA

Neural Network Configuration File: nn_bot65w.cfg

Thresholds
Magnitude: 0.8
Phase: 0.8

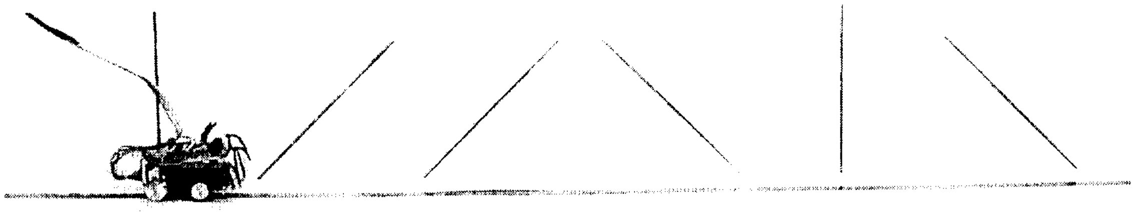
56.78

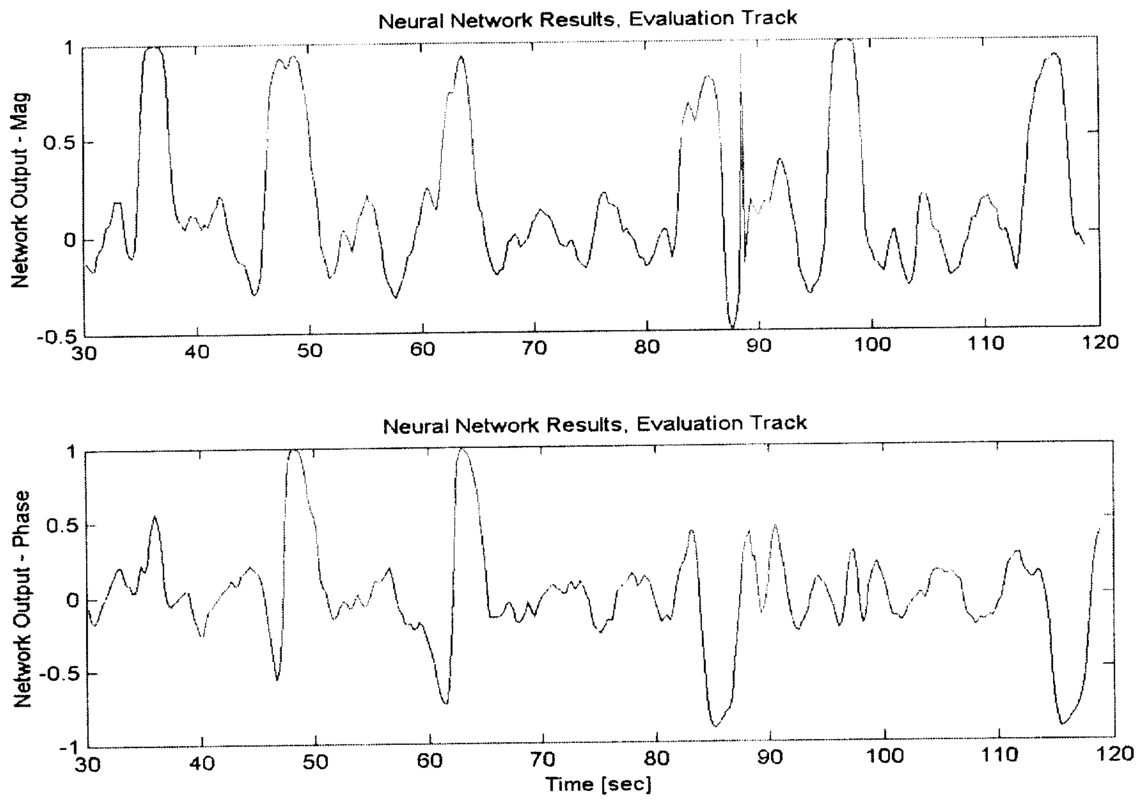
Finished evaluation 216

**Figure 8 Neural Network Analyzer**

The evolutionary designed robot was subjected to various evaluation courses including a straight track (see Figure 9) and an airfoil shaped evaluation course. The square box in Figure 8 was normally green and turned red only in the presence of a defect or crack. The dial indicator on the application provided the user with orientation of the crack or defect.

The application displays the CdS sensor and encoder measurements as the robot moves along the evaluation track. The user must select the neural network configuration file, the network results file, and the sensor output file prior to execution. The user also has the capability to set the magnitude and phase thresholds, which govern the sensitivity of the network to spikes in the data.





Neural Network Results, Evaluation Track
Network Output - Mag
Neural Network Results, Evaluation Track
Network Output - Phase
Time [sec]

**Figure 9 Evaluation Course and Evaluation Results**

Note in Figure 9 that the peaks in the magnitude (defect neuron) and phase (orientation neuron) plots correspond to the lines or "cracks" on the evaluation track. Also, the +1/-1 phase measurement coincide with the +45/-45 degree oriented "crack" on the evaluation course.

## CONCLUSIONS

In general, the defect detection network worked well given the limited time applied to the development of robust training data. The robot was reliably able to detect the presence of cracks on the straight evaluation track, but had problems with the airfoil shaped track due to the variable standoff distances of the CdS sensors when the robot was turning. The network is quite sensitive to the standoff distance of the CdS sensors. The network is also sensitive to the angle of incidence between the LED light source and the surface, which made detection of the defects difficult when turning. In order to improve the performance of the robot additional research should be performed to investigate other sensor types for defect detection and accurate standoff distances.